Controversial extension methods: CastTo and As

devblogs.microsoft.com/oldnewthing/20191227-00

December 27, 2019

Raymond Chen

You've probably had to do this in C#. You get an object, you need to cast it to some type `T` and then fetch a property that is returned as an `object`, so you have to cast *that* to some other type `U`, so you can read the destination property.

For example, you have a `ComboBoxItem`, and you put some extra data in the `Tag`.

```
void AddComboBoxItem(Thing thing)
{
    var item = new ComboBoxItem { Content = thing.Name, Tag = thing };
    someComboBox.Items.Append(item);
}

void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
    var thing = (Thing)((ComboBoxItem)((ComboBox)sender).SelectedItem)?.Tag;
    ...
}
```

In this case, when the selection changes, we ask the `ComboBox` for its currently-selected item, cast it to a `ComboBoxItem`, then get the `Tag` from it, then cast the `Tag` to the `Thing` that we were after in the first place.

In order to parse that expression, your eyes have to bounce back and forth because the casts are on the left, but the method calls and property accesses are on the right.

```
    //             6           4           2       1         3          5
    var thing = (Thing)((ComboBoxItem)((ComboBox)sender).SelectedItem)?.Tag;
```

You also have to pay attention to the parentheses, or what's more likely to be the case, you simply trust that the parentheses are in the right place.

Enter the controversial extension method `CastTo<T>`.

```
namespace ObjectExtensions
{
    static class ExtensionMethods
    {
        public static T CastTo<T>(this object o) => (T)o;
    }
}
```

With this extension method, you can write your code as a straightforward left-to-right sequence.

```
void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
    var thing = sender.CastTo<ComboBox>().SelectedItem.CastTo<ComboBoxItem>
()?.Tag.CastTo<Thing>();
    ...
}
```

You can break up the long line for readability, and the fact that there are no large spans of parentheses makes the line breaks easier to place.

```
void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
    var thing = sender
        .CastTo<ComboBox>()
        .SelectedItem
        .CastTo<ComboBoxItem>()
        ?.Tag
        .CastTo<Thing>();

    ...
}
```

Some people use the `as` operator instead of a cast, not because they actually care about the failure case (in which the result of the `as` is `null`), but because it lets them write things left-to-right.

```
void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
    //             1         2              3               4         5       6
    var thing = ((sender as ComboBox).SelectedItem as ComboBoxItem)?.Tag as Thing
    ...
}
```

This lets you read from left to right, but you still have to mind your parentheses. It looks a little prettier, but it also makes debugging harder.

You can write a similar extension method for `as`.

```
namespace ObjectExtensions
{
    static class ExtensionMethods
    {
        public static T CastTo<T>(this object o) => (T)o;
        public static T As<T>(this object o) where T : class => o as T;
    }
}
```

This lets you change the above to

```
void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
    var thing = sender
        .As<ComboBox>()
        .SelectedItem
        .As<ComboBoxItem>()
        ?.Tag
        .As<Thing>();
    ...
}
```

I suspect that like my crazy thread-switching tasks, people are going to think either that this is a really cool trick, or it's an offense against nature.

Raymond Chen

Follow